CSC104 Winter 2008 **Python Programming** **Lab – Week 6**

More Python exercises for this week:

1. **Example of using Modulus (%)**
Given two dates in the format of hours:minutes:seconds, what is the difference between them? For example, what's the time difference between 11:28:56 and 18:35:23? (The times are in the 24-hour clock. 00:00:00 is midnight).

Subtracting the individual bits is possible. But another strategy for calculating with times is to convert everything to seconds. If the values are in variables h1, m1, s1 and (for 11, 28, 56 respectively), h2, m2, s2 (for 18, 35, 23, respectively) then we can combine them with these two python statements:

```
t1 = h1*3600 + m1*60 + s1
t2 = h2*3600 + m2*60 + s2
```

Now, we can test whether t1<t2 to see which time is earlier, etc. It's very
very easy to work with in seconds. The following python code prints the time difference:

```
print 'The second time is',
sec = t2 - t1
if sec >= 3600:
   print sec/3600, 'hours and',
   sec = sec % 3600
print sec/60, 'minutes and', sec%60, 'seconds',
print 'later than the first.'
```

2. Write a python function `isEven(number)` that returns `True` if the number is even, `False` if the number is odd.

Function specification	**Usage example:**
`isEven(number)` **Input**: a number (integer) **Output**: true or false (String).	`isEven(5)` returns false `isEven(12)` returns true.

3. The following list (`days_in_month`) contains the number of days in each month for the year of 2008. Write a python function `get_days_in_month(month_num)` that takes in a month number (integer), and returns the number of days in that month.

```
days_in_month  = [31,29,31,30,31,30,31,31,30,31,30,31]
```

Function specification	**Usage example:**
`get_days_in_month (number)` **Input**: integer **Output**: integer	`get_days_in_month(1)` returns 31 `get_days_in_month(11)` returns 30

4. Write a python program that returns the average number of days per month in the `days_in_month` list. (**hint**: remember that when dividing two integers we get an integer, so if we want a decimal part, one of our numbers must be a decimal number.)

5. Let's say now we have the following two lists `student_numbers` and `student_names`. They are parallel lists because the values in the same list position correspond to each other, i.e. student with name stored in `student_name[2]` has a student number stored in `student_number[2]`.

```
student_numbers =[236435, 754678, 345245,3456677, 356325]
student_names = [‘John’, ‘Joe’, ‘Lisa’, ‘Jennifer’, ‘Mike’]
```

Write the following two functions:

Function specification	**Usage example:**

`getName(number)`
Input: a number (integer)
Output: student name (String).
	`num = 754678`
	`name = getName (num)`
	`print ‘student with id’, num, ‘has name:’, name, ‘.’`

`GetNumber(name)`
Input: a name (string)
Output: student number (string)
	`Usage example:`
	`name = ‘Jennifer’`
	`num = getNumber(name)`
	`print name, ‘has student number:’, num, ‘.’`

6. Write a function that goes through the above two lists, and prints out all student records, in the format of student names: student number.

Function specification	**Output**

`printAllRecords()`
Input: none
Output: none (just print, no return)
	`John: 236435`
	`Joe: 754678`
	`Lisa: 34524`
	`Jennifer: 3456677`
	`Mike: 356325`

7. **String Manipulation:**

- ☻ Unlike lists, for which we can change individual items, strings are “immutable” – we cannot directly substitute characters in a string with other ones. We must create new strings composed of the pieces of the string we wish to keep combined with any substitution characters.
- ☻ To slice and dice strings we have made use of the `‘:’` (colon) in class. Assume we have a string `a = ‘1234567890’.` To select the first letter in `a` we would write `a[0]`; the second letter, `a[1]` … and so on. If we want to select up to the fourth letter, we would use the colon as follows: `a[:4]` which would give us `‘1234’`. If we want the second and third letter we would write `a[1:3]` which would give us `‘23’`. Finally, if we want the 5th letter onwards we would write `a[4:]` which would give us `'567890'.`
- ☻ We add strings (and parts of strings) together using the `‘+’` (plus sign). We refer to this as string concatenation. So `a[1] + a[5:]` would be evaluated as `‘2’ + ‘67890'` or `‘267890’`. Notice that unlike the print command, which inserts a space between items that are separated with a comma, when we concatenate two strings, they are attached to each other with no spaces. We cannot concatenate a string with a number, we must first convert that number to a string using the `str(<number expression>)` function.

In the interactive python session window, try creating, cutting, splicing and rejoining a few strings.

Now, write a function that takes a string, the start position, the end position, the replacement character, replaces all characters from the start position to the end position with the replacement string, and returns the resulting string. Use the colon operator to do this.

Function specification	**Sample usage**

`stringReplace(s, startPos, endPos, replaceWith)`
Input: string `s`, the string to be manipulated;
`startPos`, the first character in the sequence to be cut out;
`endPos` the last character in the sequence to be cut out; and
`replaceWith`, the string to replace the cutout segment with.
Output: returns the resulting string.
	`a = ‘I just like to program’`
	`b = ‘love’`
	`stringReplace(a, 8, 11, b)`
	
	should return
	`‘I just love to program’`

Hint: remember that the 8th character in string `a` is `a[7]`